Thesis Title

by

Rouzbeh Paktinatkeleshteri

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Your abstract here. The abstract is not allowed to be more than 700 words and cannot include non-text content. It must also be double-spaced. The rest of the document must be at least one-and-a-half spaced.

Preface

A preface is required if you need to describe how parts of your thesis were published or co-authored, and what your contributions to these sections were. Also mention if you intend to publish parts of your thesis, or have submitted them for publication. It is also required if ethics approval was needed for any part of the thesis.

Otherwise it is optional.

See the FGSR requirements for examples of how this can look.

To the Count

For teaching me everything I need to know about math.

I think there is a world market for maybe five computers.

– Thomas J. Watson, IBM Chairman, 1943.

Acknowledgements

Put any acknowledgements here, such as to your supervisor, and supervisory committee. Remember to list funding bodies, and external scholarships. The acknowledgements can't be more than 2 pages in length.

Acknowledgements are optional, but are recommended by the FGSR.

Contents

List of Tables

List of Figures

Glossary

A Sample Acronym (ASA)

A sample acroynm description

Glossary Entry

This is a sample glossary entry.

Glossary entry descriptions can span multiple paragraphs.

Remember that glossaries are optional. The glossary implementation in this template is intended to be simple, and makes use of only one package, `glossaries`. There are more flexible, and fully-featured methods for creating glossaries than the one used here.

Chapter 1

Introduction

Here is a test reference **Knuth68:art'of'programming**. These additional lines have been added just to demonstrate the spacing for the rest of the document. Spacing will differ between the typeset main document, and typeset individual documents, as the commands to change spacing for the body of the thesis are only in the main document.

1.1 Cross-Referencing

Cross-references between child documents are possible using the `zref` package.

Text on a new page, to test top margin size.

A sample equation (1.1) follows:

$$y = \frac{1}{x^2} + 4 \tag{1.1}$$

A sample table, Table 1.1:

Non-wrapping column	**Wrapping column**
This is an ordinary column	This is a balanced-width column, where text will wrap

Table 1.1: A sample table created using the `tabulary` package

If there are many acronyms, such as A Sample Acronym (ASA), and specialized technical terms, consider adding a glossary. Sample glossary entry, and acroynm (ASA) descriptions are provided above.

Chapter 2

Main Chapter

SIMD intructions allow modern processor to apply the same Intruction on multiple data at the ame time. The performance improvement gained from these intructions is so considerable(?) that made compiler specialists to explore different ways to exploite SIMD intructions. Auto-vectorization [cite] is a compiler tranformation that is proposed for this purpose. Implemented by almost all current compilers, auto-vectorization looks for possibility of using SIMD instructions (also called vector instructions) in the program and replaces scalar code (code that is made of simple instructions) with vector instructions wherever possible.

Since most of the execution time of a program is spent on loops, vectorization is typically applied on loops. Famous Compilers have optimization passes (such as slp-vectorizer in clang ... [cite, more examples]) that vectorize loops body. In spite of loop-vectorization there has been efforts to vectorize other structres such as functions [cite] as well however, the focus of research in this area is on loops.

A huge amount of work has been done to improve codes using vectorization [cite] however, the transformation needs the code to meet certain requirements which if not met, would result in invalid code produced by the compilers. Furthuremore, replacing scalar code with vector is not always benefitial. In some situations(?) scalar code can provide better performace in comparison to vector code. In response to these two problems with vectorization, compilers come with an analysis pass to check both legality of the tranformation and it's

profitability.

One of the larget onbstacles for vectorization is control flow divergence. Existence of branches (such as if-then-else statements or swithch case statements) causes the program to take different paths during execution time based on some conditions inside the code that could change dynamically. This is called divergence in the control flow of the program. Having divergence in the code, vectorization can not be simply applied, as different iterations of the loop might take different paths and as a result disabling the compiler to replace instructions with SIMD ones.

Chapter 3

Conclusion

Referring back to the introduction (Section 1.1), we see that cross-references between files are correctly handled when the files are compiled separately, and when the main document is compiled. When the main document is compiled, cross-references are hyperlinked. The values of the cross-references will change between the two compilation scenarios, however. (Each chapter, compiled on its own, becomes "Chapter 1".)

Caution: For cross-references to work, when files are compiled separately, the referenced file must be compiled at least once before the referring file is compiled.

Appendix A

Background Material

Material in an appendix.

We plot an equation in figure **??**.

,,